(12) DEMANDE INTERNATIONALE PUBLIÉE EN VERTU DU TRAITÉ DE COOPÉRATION EN MATIÈRE DE BREVETS (PCT)

(19) Organisation Mondiale de la Propriété Intellectuelle
Bureau international

(43) Date de la publication internationale
28 juillet 2005 (28.07.2005)

PCT

(10) Numéro de publication internationale
WO 2005/069122 A2

(51) Classification internationale des brevets[7] : G06F 7/72

(21) Numéro de la demande internationale :
PCT/EP2004/053472

(22) Date de dépôt international :
14 décembre 2004 (14.12.2004)

(25) Langue de dépôt : français

(26) Langue de publication : français

(30) Données relatives à la priorité :
0314959 19 décembre 2003 (19.12.2003) FR

(71) Déposant *(pour tous les États désignés sauf US)* : **GEMPLUS** [FR/FR]; Avenue du Pic de Bertagne Parc, d'activités de Gémenos, F-13420 Gemenos (FR).

(72) Inventeur; et
(75) Inventeur/Déposant *(pour US seulement)* : **CHEVALLIER-MAMES, Benoit** [FR/FR]; La Sardanne Résidence Les Brayes, F-13260 Cassis (FR).

(81) États désignés *(sauf indication contraire, pour tout titre de protection nationale disponible)* : AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NA, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW.

(84) États désignés *(sauf indication contraire, pour tout titre de protection régionale disponible)* : ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), eurasien (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), européen (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IS, IT, LT, LU, MC, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Publiée :
— *sans rapport de recherche internationale, sera republiée dès réception de ce rapport*

En ce qui concerne les codes à deux lettres et autres abréviations, se référer aux "Notes explicatives relatives aux codes et abréviations" figurant au début de chaque numéro ordinaire de la Gazette du PCT.

(54) **Title:** CRYPTOGRAPHIC METHOD FOR MODULAR EXPONENTIATION, PROTECTED AGAINST DPA-TYPE ATTACKS

(54) **Titre :** PROCEDE CRYPTOGRAPHIQUE D'EXPONENTIATION MODULAIRE PROTEGE CONTRE LES ATTAQUES DE TYPE DPA

(57) **Abstract:** The invention relates to the protection of cryptographic methods against DPA-type covert channel attacks and, in particular, to a cryptographic method during which an x^d-type modular exponentiation is performed, wherein d is a whole number exponent of m+1 bits, consisting in: scanning the d bits from left to right in a loop subscripted by i varying between m and 0; and, with each revolution of rank i, calculating and saving an updated partial result equal to x^b(i) in an accumulator (R0), b(i) being the most significant m-i+1 bits of exponent d (b (i) = $d_{m->i}$). According to the invention, at the end of a revolution of randomly-selected rank i(j) (i = i(0)), a randomisation step E1 is performed, consisting in subtracting a random number z (z= b(i(j)), z = b (i(j)) .2^{τ}, z = u) from part of the d bits that have not yet been used ($d_{i-1->0}$) in the method. Subsequently, once the d bits modified by randomisation step E1 have been used, a consolidation step E2 is performed, consisting in saving (R0 <- R1xR0), in the accumulator (R0), the result of the multiplication of the contents of the accumulator (x^b(i)) by a number that is a function of x^z stored in a registry (R1).

(57) **Abrégé :** Dans le domaine de la protection des procédés cryptographiques contre les attaques à canaux cachés de type DPA, l'invention concerne un procédé cryptographique au cours duquel on réalise une exponentiation modulaire de type x^d, avec d un exposant entier de m+1 bits, en balayant les bits de d de gauche à droite dans une boucle indicée par i variant de m à 0 et en calculant et en mémorisant dans un accumulateur (RO), à chaque tour de rang i, un résultat partiel actualisé égal à x^b(i), b(i) étant les m-i+1 bits de poids les plus forts de l'exposant d (b (i) = $d_{m->i}$) .Selon l'invention, à la fin d'un tour de rang i(j) (i = i(0)) choisi aléatoirement, on réalise une étape E1 de randomisation au cours de laquelle E1: on soustrait un nombre z (z= b(i(j)), z = b (i(j)) .2^{τ}, z = u) aléatoire à une partie des bits de d non encore utilisés ($d_{i-1->0}$) dans le procédé puis, après avoir utilisé les bits de d modifiés par l'étape de randomisation E1, on réalise une étape de consolidation E2 au cours de laquelle: E2 : on mémorise (R0 <- R1xR0) dans l'accumulateur (RO) le résultat de la multiplication du contenu de l'accumulateur (x^b(i)) par un nombre fonction de x^z mémorisé dans un registre (R1).

WO 2005/069122 A2

PROCEDE CRYPTOGRAPHIQUE D'EXPONENTIATION MODULAIRE PROTEGE CONTRE LES ATTAQUES DE TYPE DPA

Dans le domaine de la protection des algorithmes cryptographiques contre les attaques DPA, l'invention concerne un procédé au cours duquel on réalise une exponentiation modulaire de type x^d, avec d un exposant entier de m+1 bits, en balayant les bits de d de gauche à droite dans une boucle indicée par i variant de m à 0 et en calculant et en mémorisant dans un accumulateur (R0), à chaque tour de rang i, un résultat partiel actualisé égal à x^b(i). b(i) correspond aux m-i+1 bits de poids les plus forts de l'exposant d : $b(i) = d_{m->i}$. Le nombre constitué des bits de poids j à k de d est défini par : $d_{k->j} = (d_k, \ldots, d_j)_2 = \sum_{i=j}^{k} d_i . 2^{(i-j)}$.

L'exponentiation modulaire est une des opérations élémentaires utilisés dans de nombreux cryptosystèmes, tels que les cryptosystèmes RSA (Rivest, Shamir and Adleman) ou les cryptosystèmes DH (Diffie and Hellman). Pour de telles applications, x est par exemple un message à chiffrer ou déchiffrer, à signer ou à authentifier, et d est par exemple une clé publique, une clé secrète, ou une partie d'une telle clé.

Depuis l'invention de la cryptographie à clé publique par Diffie et Hellman, de nombreux cryptosystèmes à clé publique ont été proposés. Parmi ceux qui résistent à l'analyse cryptographique, le cryptosystème RSA est sans aucun doute le plus largement utilisé. Sa sécurité intrinsèque réside dans la difficulté de factoriser des grands nombres entiers. En dépit d'intensives recherches, le problème de la factorisation est encore considéré comme un important problème, rendant le cryptosystème RSA

sûr pour des applications sensibles comme par exemple le chiffrement de données ou la signature digitale.

Aussi, plutôt que de tenter de casser l'algorithme RSA à un niveau mathématique, les cryptographes se sont intéressés aux implémentations concrètes des cryptosystèmes RSA. Ceci a conduit à l'essor des attaques par fautes et des attaques à canaux cachés, visant à découvrir notamment des informations confidentielles (comme par exemple des clés ou des parties des clés) manipulées au cours de l'une ou de l'autre des étapes mises en œuvre par le dispositif de calcul exécutant une opération cryptographique.

Les attaques à canaux cachés les plus connues sont dites simples ou différentielles. On entend par attaque à canal caché simple (SPA) ou différentielle (DPA), une attaque basée sur la mesure d'une grandeur physique depuis l'extérieur du dispositif, dont l'analyse directe (attaque simple SPA) ou l'analyse selon une méthode statistique (attaque différentielle DPA) permet de découvrir des informations manipulées dans le dispositif. Ces attaques ont notamment été dévoilées par Paul Kocher (Advances in Cryptology - CRYPTO'99, vol. 1666 of Lecture Notes in Computer Science, pp.388-397. Springer-Verlag, 1999).

Parmi les grandeurs physiques qui peuvent être exploitées à ces fins, on peut citer le temps d'exécution, la consommation en courant, le champ électromagnétique rayonné par la partie du composant utilisée pour exécuter le calcul, etc. Ces attaques sont basées sur le fait que, au cours de l'exécution d'un algorithme, la manipulation d'un bit, c'est à dire son utilisation par une instruction particulière, laisse une empreinte

particulière sur la grandeur physique considérée, selon la valeur de ce bit et / ou selon l'instruction.

Il existe deux familles d'implémentations des algorithmes d'exponentiation permettant d'évaluer la valeur de y = x^d mod N : les implémentations dites de droite à gauche et les implémentations dites de gauche à droite.

Dans les implémentations de gauche à droite, on balaye les bits de l'exposant depuis le bit de poids le plus fort jusqu'au bit de poids le plus faible. Dans cette deuxième famille d'algorithmes d'exponentiation, est notamment connu l'algorithme SAM (pour Square And Multiply ou élever au carré et multiplier) et ses variantes telles que les algorithmes à fenêtre glissante. Par rapport aux algorithmes dits de droite à gauche, les algorithmes de gauche à droite nécessitent moins de mémoire et permettent l'utilisation de puissances x^i précalculées pour accélérer le calcul de y. Tous les algorithmes de gauche à droite ont en commun l'utilisation d'un accumulateur (ou registre) qu'on actualise tout au long du calcul pour mémoriser la valeur de $x^{d_{m->i}} \bmod N$ pour des valeurs décroissantes de i jusqu'à ce que l'accumulateur contienne la valeur finale $y = x^{d_{m->0}} = x^d \bmod N$. $d_{k->j}$ est le mot constitué des bits de poids j à k de d.

Le principe général de l'algorithme SAM est le suivant. On note $d = (d_m, \ldots, d_0)_2 = \sum_{i=0}^{m} d_i . 2^i$, la représentation binaire de l'exposant d, avec $d_i \in \{0, 1\}$, le bit de poids i de d. Pour chaque bit de d, l'algorithme SAM mémorise dans un accumulateur (registre R0) un résultat actualisé calculé à partir de la relation de récurrence

$x^{d_{m->i}} = (x^{d_{m->i+1}})^2 * x^{d_i}$, avec $x^{d_{m->m}} = x^{d_m}$, ce qui est résumé par l'algorithme ci-dessous :

Entrée : x, $d = (d_m, \ldots, d_0)_2$

Sortie : $y = x^d \bmod N$

R0 <- 1 ; R2 <- x, i <- m

tant que $i \geq 0$, faire :

R0 <- R0×R0 mod N

si $d_i = 1$ alors R0 <- R0×R2 mod N

i <- i-1

fin tant que

retourner R0

R0 <- x signifie que l'on mémorise dans le registre R0 la valeur de x. R0×R0 signifie que l'on réalise une mise au carré du contenu du registre R0. R0×R2 signifie qu'on réalise le produit du contenu du registre R0 par le contenu du registre R2. Enfin, $d_{i->j}$ fait référence aux bits de rang j à i de d.

Pour prévenir les attaques d'implémentation, il est connu qu'il faut rendre les algorithmes aléatoires. Dans le cas du cryptosystème RSA, on connaît actuellement deux types de contre-mesures pour rendre aléatoire le calcul de $y = x^d \bmod N$.

Le premier type de contre-mesure consiste à rendre aléatoire les données d'entrée de l'algorithme.

Un premier exemple de cette première contre-mesure consiste à rendre aléatoire la donnée x avant de réaliser l'exponentiation modulaire, en ajoutant à x un terme aléatoire et à faire les calculs modulo $2^k N$, avant un modulo N final :

$\overline{x}$ <- x + r1.N, avec r1 un nombre aléatoire de k-bits

et à faire les calculs modulo (2^k).N, avant une réduction modulo N finale. Cette première contre-mesure, décrite par P. Kocher, présente l'avantage d'être indépendante de l'algorithme d'exponentiation.

Un deuxième exemple de cette première contre-mesure consiste à rendre aléatoire l'exposant d avant de réaliser l'exponentiation modulaire, en lui ajoutant un terme aléatoire :

$\overline{d} \leftarrow d + r2.\phi(N)$, r2 un nombre aléatoire de k-bits.

Le plus souvent, ces deux solutions sont combinées pour réaliser l'opération $y = \overline{y} \bmod N$ avec $\overline{y} = \overline{x}^{\overline{d}} \bmod (2^k.N)$.

Un troisième exemple de cette première contre-mesure utilisée seule par exemple lorsque x est le résultat d'un formatage probabiliste (par exemple à l'aide de la fonction PSS ou Probabilistic Signature Scheme), car dans ce cas, x est déjà masqué et on calcule directement $y = x^{\overline{d}} \bmod N$ avec $\overline{d} = d + r2.\phi(N)$ avec r2 aléatoire.

Malheureusement, une telle randomisation de l'exposant d est limitée à des implémentations particulières, appelées implémentations CRT, du cryptosystème RSA car la valeur de la constante d'Euler $\phi(N)$ n'est généralement pas connue de l'algorithme d'exponentiation privé dans sa version standard (c'est-à-dire non CRT).

La deuxième contre-mesure consiste à rendre l'algorithme d'exponentiation lui-même aléatoire. La meilleure mise en pratique de la 2ème contre-mesure est l'algorithme MIST de Walter. L'algorithme MIST génère de manière aléatoire une nouvelle chaîne d'addition pour l'exposant d pour réaliser x^d mod N. Pour minimiser le nombre de registres, la chaîne d'addition est réalisée à la volée

par l'intermédiaire d'une adaptation d'un algorithme d'exponentiation basé sur des chaînes de divisions. Un autre exemple est une version améliorée d'un algorithme à fenêtre glissante (voir Kouichi Itoh, Jun Yajima, Masahiko Takenaka and Naoya Torii. DPA countermeasures by improving the window method CHES 2002, volume 2523 of Lecture Notes in computer Science, pages 303-317, Springer Verlag 2002). Comparé à la première contre-mesure, ceci permet de rendre aléatoire l'exponentiation sans avoir besoin de connaître $\phi(N)$ mais nécessite un algorithme de division sécurisé pour calculer les chaînes de divisions et cause des soucis de gestion des calculs non négligeables.

L'invention propose un nouveau procédé pour rendre aléatoire l'exécution d'une exponentiation modulaire, dans le but de prévenir les attaques différentielles (DPA), présentant les avantages des deux contre-mesures connues : comme dans la première contre-mesure, le procédé selon l'invention n'impose pas d'algorithme d'exponentiation particulier et s'applique à tout algorithme d'exponentiation, et comme dans la deuxième contre-mesure, dans l'invention, l'algorithme lui-même est rendu aléatoire, et non plus seulement les données qu'il manipule. Ainsi, l'algorithme n'a pas besoin de connaître $\phi(N)$ et / ou la clé publique e dans une exponentiation RSA (la clé e est souvent indisponible à l'algorithme de signature ou de déchiffrement).

Le procédé selon l'invention introduit le concept de exponentiation auto-aléatoire, signifiant que l'exposant d est utilisé lui-même comme une source additionnelle d'aléatoirité dans le processus d'exponentiation.

Ainsi, l'invention concerne un procédé cryptographique au cours duquel on réalise une exponentiation modulaire de

type x^d, avec d un exposant entier de m+1 bits, en balayant les bits de d de gauche à droite dans une boucle indicée par i décrémenté de m à 0 par pas de 1 et en calculant et en mémorisant dans un accumulateur, à chaque tour de rang i, un résultat partiel actualisé égal à x^b(i), b(i) étant les m-i+1 bits de poids les plus forts de l'exposant d.

Le procédé selon l'invention est caractérisé en ce que :

- à la fin d'un tour de rang i(j) (i = i(0)) choisi aléatoirement, on réalise une étape E1 de randomisation au cours de laquelle :

  E1 : on soustrait un nombre z (z= b(i(j)), z = b(i(j)).2^{τ}, z = u) aléatoire à une partie des bits de d non encore utilisés ($d_{i-1->0}$) dans le procédé

- puis, après avoir utilisé les bits de d modifiés par l'étape de randomisation E1, on réalise une étape de consolidation E2 au cours de laquelle :

  E2 : on mémorise (R0 <- R1×R0) dans l'accumulateur (R0) le résultat de la multiplication du contenu de l'accumulateur (x^b(i)) par un nombre fonction de x^z mémorisé dans un registre (R1).

D'un point de vue pratique, au cours de l'étape E1, le nombre z est soustrait au contenu d'un registre dans lequel est mémorisé initialement l'exposant d, et le résultat de la soustraction est mémorisé dans le même registre, puis on continue de balayer les bits de b.

Pour que le résultat de l'exponentiation x^d mod N soit correct à la fin du procédé, l'étape de randomisation E1, ne doit pas modifier les bits de d déjà utilisés dans le calcul (on rappelle que le procédé utilise un algorithme de gauche à droite). L'index i(j) auquel on réalise la

randomisation E1, choisi aléatoirement, doit donc être choisi tel que les m-i(j)+1 bits de poids les plus forts du registre contenant initialement l'exposant d restent inchangés lors de l'étape E1. On appellera par la suite cette condition une condition de "consistance".

L'idée essentielle de l'invention st ainsi d'utiliser un découpage du calcul de x^d mod N de la forme : x^d = x^(d-z)*x^z (décrit dans la demande de brevet français n° 02 04117 (n° à confirmer) avec z un nombre aléatoire utilisé comme moyen de masquage de l'exposant d. On choisit de préférence des valeurs de z appropriées telles que x^z puisse être obtenu facilement à partir de x^b déjà calculé par ailleurs au cours du procédé. A noter qu'un choix totalement aléatoire de z entraîne un quasi doublement du temps de calcul.

Le procédé selon l'invention s'applique indépendamment de l'algorithme d'exponentiation de gauche à droite. Par ailleurs, le rang i(j) auquel on réalise l'étape E1 est choisi aléatoire donc le procédé lui-même est aléatoire, et non pas seulement les données qu'il manipule.

Le procédé selon l'invention est également efficace en termes de place (il ne nécessite qu'un seul registre de calcul supplémentaire) et en terme de temps de calcul, comme on le verra mieux par la suite dans l'exemple de l'algorithme SAM.

Le procédé selon l'invention est encore facile à mettre en œuvre quel que soit l'algorithme auquel il est appliqué : il ne repose sur aucune propriété de groupe et sa mise en œuvre ne nécessite pas de connaître au

préalable l'ordre du groupe dans lequel l'exponentiation est réalisée.

Enfin, le procédé selon l'invention peut être utilisé conjointement avec d'autres mesures de protection des algorithmes, comme par exemple les contre mesures dévoilées par P. Kocher et rappelées précédemment.

L'étape E1 de randomisation peut être réalisée une seule fois au cours du procédé. L'étape E1 peut également être réalisée plusieurs fois, à la fin de différents tours de rang i(j) (c'est-à-dire au rang i = i(0), puis au rang i = i(1), …, puis enfin au rang i = i(f)) choisis aléatoirement entre 0 et m. L'idée est ici d'améliorer encore la sécurité du procédé en utilisant la relation :

$$x^d = x^{(d-z1-z2-\ldots-zf)} \times x^{z1} \times x^{z2} \times \ldots \times x^{zf}$$
$$= x^{(((d-z1)-z2)-\ldots-zf)} \times ((x^{z1}) \times x^{z2}) \times \ldots \times x^{zf}$$

On peut choisir au début du procédé le ou les rangs i(j) aléatoires auxquels on réalise une randomisation E1. Par exemple, au début du procédé, on détermine un ensemble prédéfini {i(0), i(1), …, i(f)} de f+1 (f étant aléatoire ou non) valeurs de l'index i pour lesquelles on souhaite réaliser une randomisation E1. Dans ce cas, à chaque tour, on décide de réaliser ou non une randomisation E1 selon que l'index courant i fait partie ou non de l'ensemble prédéfini.

On peut aussi choisir aléatoirement au début de chaque tour i de réaliser ou non l'étape E1 de randomisation. Dans ce cas, on utilise par exemple une variable booléenne ρ, tirée aléatoirement à la fin de chaque tour d'index i.

Différents modes de réalisation de l'invention vont maintenant être décrits, qui diffèrent l'un de l'autre essentiellement par le mode de réalisation de l'étape E1, et notamment par le choix de z et par le choix de la partie de d à laquelle on soustrait z.

Selon un premier mode de réalisation, on choisit de réaliser l'étape de consolidation une seule fois à la fin du procédé. Ceci impose de soustraire z systématiquement aux bits de poids les plus faibles de l'exposant d, de sorte à obtenir un résultat correct à la fin du procédé.

Selon une première variante de ce mode de réalisation, on choisit $z = b(i(j)) = d_{m->i(j)}$ pour un nombre aléatoire i(j) choisi et, au cours de l'étape de randomisation E1, on soustrait b(i(j)) à d, c'est-à-dire aux bits de poids les plus faibles de d.

Le choix z = b(i(j)) est particulièrement avantageux puisque $x^{b(i(j))} = x^{d_{m->i(j)}}$ est déjà disponible dans l'accumulateur à la fin du tour i(j) et n'a donc pas besoin d'être calculé. La variable i(j) est choisie telle que les bits de poids i(j) à m du nombre d-b(i(j)) soient égaux aux bits de poids i(j) du nombre d, de sorte que les m-i(j)+1 premiers tours du calcul de x^d sont identiques aux m-i(j)+1 premiers tours du calcul de $x^{(d-b(i(j)))}$ (condition de consistance). A la fin du tour i(j), on calcule d-z = d-b(i(j)) et on mémorise le contenu de l'accumulateur x^b dans le registre (E1).

Dans un exemple particulier, une variable booléenne ρ est utilisée pour déterminer, à la fin de chaque tour d'index i, si on réalise ou non une randomisation. Si ρ prend une valeur active, alors on réalise l'étape E1 : on remplace le nombre d par le nombre d-b(i(j)) et on mémorise x^b(i(j)).

Comme dans l'algorithme de gauche à droite classique, l'accumulateur R0 est utilisé pour conserver la valeur de $x^{d_{m->i}}$, à chaque tour d'index i. On utilise le registre R1 pour conserver le produit : $\prod_j x^{d_{m->i(j)}}$.

Le tout appliqué à l'algorithme SAM connu, on obtient le l'algorithme I suivant :

```
Entrée : x, d = (d_m, ..., d_0)_2
Sortie : y = x^d mod N
    R0 <- 1 ; R1 <- 1 ; R2 <- x ; i <- m
    tant que i ≥ 0, faire :
        R0 <- R0×R0 mod N
        si d_i = 1 alors R0 <- R0×R2 mod N
        ρ <- R{0, 1}
        si ((ρ = 1) ET d_{i-1->0} ≥ d_{m->i} alors
            d ← d - d_{m->i}
            R1 <- R1×R0 mod N
        fin si
        i <- i-1
    fin tant que
    R0 <- R0×R1 mod N
retourner R0
```

ρ <- R{0, 1} signifie que l'on choisit la valeur de ρ aléatoirement dans l'ensemble {0, 1}. ρ est ainsi une variable booléenne aléatoire.

L'étape de randomisation E1 (d <- d - $d_{m->i(j)}$; R1 <- R1×R0 mod N) est réalisée uniquement si ρ = 1 (c'est-à-

dire si une randomisation doit être réalisée) et si $d_{i(j)-1->0} \geq d_{m->i(j)}$.

La condition $d_{i(j)-1->0} \geq d_{m->i(j)}$ signifie que les bits de poids 0 à i-1 de d sont supérieurs à b(i(j)), b(i(j)) étant égal aux bits de poids i(j) à m de d. Ceci permet de garantir que les m-i+1 bits de poids les plus forts de d-b(i(j)) sont identiques aux m-i+1 bits de poids forts de d, et donc que les m-i+1 premiers tours du calcul de x^d sont identiques aux m-i+1 premiers tours du calcul de x^(d-b(i(j))).

La condition de "consistance" ($d_{i(j)-1->0} \geq d_{m->i(j)}$) implique que seulement les bits de poids faibles de l'exposant d sont rendus aléatoires. De plus, on remarque que l'étape de randomisation d <- d - $d_{m->i(j)}$ modifie uniquement les (m-i(j)+1) bits les moins significatifs de d.

A noter que, dans l'algorithme I, comme à l'itération i = i(j) l'étape de mise à jour d <- d - $d_{m->i}$ ne modifie pas les (m-i+1) bits les plus significatifs de d, cette étape peut être remplacée par l'étape équivalente :

$d_{i-1->0} <- d_{i-1->0} - d_{m->i}$.

Selon une deuxième variante du premier mode de réalisation, on choisit z égal g.b(i), avec g un nombre aléatoire tel que $d_{i(j)-1->0} \geq g.d_{m->i(j)}$. On utilise dans ce cas la relation $x^d = x^{(d-z)}.x^z = x^{(d-g.b(i))}.(x^{b(i)})^g$. et, d'un point de vue pratique, pour réaliser une randomisation E1 à la fin du tour d'index i(j):

- on calcule z = g.b(i) et on soustrait le résultat à l'exposant d,

- le registre R1 est mis à jour en multipliant son contenu par le contenu de l'accumulateur (x^b(i)) exposé à la puissance g. Ce qui concrètement peut être réalisé par l'instruction R1 <- R1×R0^g mod N.

On choisit de préférence $g = 2^{\tau}$, τ étant un nombre entier aléatoire. Ceci simplifie considérablement les calculs car le calcul de $g.b(i) = g.d_{m->i(j)}$ revient à un simple décalage de bits et l'évaluation de (x^b(i))^g mod N revient à réaliser le calcul de τ carrés.

Puisque multiplier par 2^{τ} revient à un décalage de bits, l'instruction $d <- d - 2^{\tau}.d_{m->i}$ qui calcule d-g.b(i) peut être remplacée par $d_{m->\tau} <- d_{m->\tau} - d_{m->i}$ ou mieux, par l'instruction équivalente $d_{i-1->\tau} <- d_{i-1->\tau} - d_{m->i}$.

De plus, comme pour les autres modes de réalisation, on doit vérifier que à l'itération i=i(j), $d_{i-1->0} \geq 2^{\tau}.d_{m->i}$. Cette condition de consistance peut être remplacée par un test équivalent mais plus efficace : $d_{i-1->\tau} \geq d_{m-i}$.

De préférence, on choisit τ aléatoire dans l'ensemble {0, …, T}. La borne T est choisie comme le meilleur compromis entre la randomisation des bits les plus significatifs de d et l'efficacité (en terme de temps de calcul notamment) du calcul des τ carrés.

Dans l'exemple particulier de l'algorithme SAM, on obtient finalement l'algorithme I' suivant.

```
Entrée : x, d = (d_m, ..., d_0)_2
  Sortie : y = x^d mod N
      R0 <- 1 ; R1 <- 1 ; R2 <- x ; i <- m
      tant que i ≥ 0, faire :
          R0 <- R0×R0 mod N
          si d_i = 1 alors R0 <- R0×R2 mod N
          ρ <- R{0, 1} ; τ <- R{0, ..., T}
          si ((ρ = 1) ET (d_{i-1→τ} ≥ d_{m-i})) alors
              d_{i-1→τ} ← d_{i-1→τ} − d_{m->i}
              R3 <- R0
              tant que (τ > 0) faire
                  R3 <- R3^2 mod N ; τ <- τ-1
              fin tant que
              R1 <- R1×R3 mod N
          fin si
          i <- i-1
      fin tant que
      R0 <- R0×R1 mod N
  retourner R0
```

Un avantage de l'algorithme I' est qu'on randomise en partie la moitié supérieure de d et qu'on ajoute en conséquence de l'entropie (c'est-à-dire de l'aléatoirité). Par contre, un registre supplémentaire R3 est nécessaire pour calculer $R0^{2^\tau}$.

Les algorithmes I et I' peuvent être suffisants pour protéger les exposants dans certains cas. Par exemple, en raison de sa construction, le cryptosystème RSA dévoile toujours la moitié la plus significative de l'exposant d privé si l'exposant public correspondant est petit. Rendre aléatoire les bits de poids forts de d

n'apporterait donc aucune protection pour un tel algorithme.

Toutefois, pour d'autres algorithmes et dans d'autres situations, rendre aléatoire tous les bits de d apporterait une sécurité supplémentaire.

Pour cela, on propose, dans un deuxième mode de réalisation, de choisir $z = b(i(j)) = d_{m->i(j)}$ pour un nombre aléatoire i(j) et, au cours de l'étape E1, on soustrait b(i) non plus à d, mais à une partie des bits de d correspondant aux bits de d de poids i(j)-c(j) à i(j)-1, c(j) étant un nombre entier tel que $i(j) \geq c(j) \geq 0$. Ceci peut être exprimé par l'instruction suivante :

$$d_{m->i(j)-c(j)} <- d_{m->i(j)-c(j)} - d_{m->i(j)}$$

Préférentiellement, comme lors de l'étape E1 d'une randomisation au rang i(j), on modifie les bits de poids i(j) - c(j) à i(j) - 1 de d et on choisit de ne réaliser qu'une seule randomisation à la fois, et on choisit de réaliser une étape de consolidation à la fin du rang utilisant le dernier bit de d modifié lors de l'étape de randomisation E1 précédente (et non plus à la fin du procédé), c'est à dire après l'évaluation du résultat partiel $x^{(d_{m->i(j)-c(j)})}$ mod N.

Ceci revient à imposer la condition $i(j+1) \leq i(j) - c(j)$, i(j+1) étant l'index de la randomisation suivante. Ceci permet de ne pas utiliser de registres supplémentaires pour mémoriser les bits de l'exposant qui ont été modifiés lors d'une randomisation précédente. Egalement, on choisit $i(j) - c(j) < 0$, de sorte que i(j) - c(j) puisse être utilisé pour définir le rang d'un bit de d pour le calcul de $x^{(d_{m->i(j)-c(j)})}$ mod N.

Ces deux conditions peuvent être concrétisées par l'utilisation d'un sémaphore booléen σ qui indique si une mise à jour est autorisée ou pas : σ a une valeur inactive tant que $i \geq i(j) - c(j)$ et est activée lorsque $i < i(j) - c(j)$. Elle devient de plus inutilisable dès que $i(j) - c(j) < 0$.

Les $(m-i(j)+1)$ bits de poids les plus forts de d restent inchangés lors de l'étape de randomisation si (condition de consistance) :

$$d_{i(j)-1 \to i(j)-c(j)} \geq d_{m \to i(j)}$$

$$(i(j) - 1) - (i(j) - c(j)) \geq m - i(j) \Leftrightarrow c(j) \geq m - i(j) + 1$$

Selon une première variante du deuxième mode de réalisation, on choisit $c(j)$ égal à $m-i(j)+1$. Avec la condition $i(j) \geq c(j) \geq 0$, la condition $c(j) \geq m - i(j) + 1$ est vérifiée si $2.i(j) \geq m+1$.

L'algorithme SAM modifié selon cette variante peut alors s'écrire (algorithme II) :

```
Entrée : x, d = (d_m, ..., d_0)_2
Sortie : y = x^d mod N
    R0 <- 1 ; R1 <- 1 ; R2 <- x ;
    i <- m ; c <- -1 ; σ <- 1
    tant que i ≥ 0, faire :
        R0 <- R0×R0 mod N
        si d_i = 1 alors R0 <- R0×R2 mod N  fin si
        si (2i ≥ m+1) ET (σ = 1)  alors c <- m-i+1
                                  sinon σ = 0
        fin si
        ρ <- R{0, 1}
        ε <- ρ ET (d_{i-1 -> i-c} ≥ d_{m->i}) ET σ
        si ε = 1 alors
              R1 <- R0 ; σ <- 0
              d_{i-1 -> i-c} <- d_{i-1 -> i-c} - d_{m -> i}
        fin si
        si c = 0 alors  R0 <- R0×R1 mod N ; σ <- 1
        fin si
        c <- c-1 ; i <- i-1
    fin tant que
retourner R0
```

On notera que l'algorithme I correspond à l'algorithme II dans le cas où $c(j) = i(j)$ pour tout j. On notera également que dans l'algorithme II, la condition de consistance ($d_{i(j)-1 \to i(j)-c(j)} \geq d_{m \to i(j)}$) est satisfaite durant la première partie de l'algorithme, en considérant de manière approximative que $d_{i(j)-1 \to i(j)-c(j)}$ et $d_{m \to i(j)}$ sont des nombres aléatoires de $(m-i(j)+1)$ bits. On notera, dans cet algorithme, que tous les bits de l'exposant sont randomisés.

Selon une deuxième variante du deuxième mode de réalisation, on choisit c(j) aléatoire et compris entre i(j) et m-i(j)+1.

On a vu précédemment que la condition $c(j) \geq m - i(j) + 1$ doit être vérifiée. En posant $c(j) = m - i(j) + 1 + \nu(j)$, on doit donc vérifier $\nu(j) \geq 0$. Par ailleurs, avec la condition $i(j) \geq c(j) \geq 0$, il vient $2.i(j) \geq m+1+\nu(j)$. Donc la plus grande valeur possible pour $\nu(j)$ est $2.i(j)-m-1$ et donc, comme $\nu(j) \geq 0$, le paramètre $c(j) = m-i(j)+1+\nu(j)$ peut prendre toute valeur dans l'ensemble {m-i(j)+1, …, i(j)}. On peut alors généraliser l'algorithme II en choisissant c(j) aléatoire dans l'ensemble {m-i(j)+1, …, i(j)}.

Dans l'exemple particulier de l'algorithme II cela revient à remplacer l'instruction :

si (2i ≥ m+1) ET (σ = 1) alors c <- m-i+1

par l'instruction :

si (2i ≥ m+1) ET (σ = 1) alors c <- R{m-i+1, i}

ce qui donne l'algorithme III suivant :

```
Entrée : x, d = (d_m, ..., d_0)_2
  Sortie : y = x^d mod N
      R0 <- 1 ; R1 <- 1 ; R2 <- x ;
      i <- m ; c <- -1 ; σ <- 1
      tant que i ≥ 0, faire :
          R0 <- R0×R0 mod N
          si d_i = 1 alors R0 <- R0×R2 mod N  fin si
          si (2i ≥ m+1) ET (σ = 1)
                  alors c <- R{m-i+1, i}
                  sinon σ = 0
          fin si
          ρ <- R{0, 1}
          ε <- ρ ET (d_{i-1 -> i-c} ≥ d_{m->i}) ET σ
          si ε = 1 alors
                R1 <- R0 ; σ <- 0
                d_{i-1 -> i-c} <- d_{i-1 -> i-c} − d_{m -> i}
          fin si
          si c = 0 alors  R0 <- R0×R1 mod N ; σ <- 1
          fin si
          c <- c-1 ; i <- i-1
      fin tant que
  retourner R0
```

Une valeur plus grande pour $\nu(j)$ augmente la probabilité de succès pour la condition de consistance (et donc pour le choix d'une randomisation). D'un autre côté, cela réduit également les valeurs possibles de l'indexe i vérifiant la condition $2.i(j) \geq m+1+\nu(j)$.

La fréquence d'occurrence de la valeur $\rho = 0$ de la variable booléenne ρ est un paramètre du procédé permettant de choisir le meilleur compromis entre

performance et sécurité, en fonction de l'application envisagée : plus on réalise d'étapes de randomisation, plus on pénalise le temps de calcul global ; inversement, moins on réalise d'étapes de randomisation, plus on facilite les attaques par recherche exhaustive.

Un bon moyen pour minimiser le coût des opérations additionnelles consiste à modifier légèrement le générateur de nombres aléatoires produisant le nombre ρ de sorte que, lorsque le poids de Hamming de d-z (z peut avoir différentes valeurs en fonction de b(i), selon le mode de réalisation envisagé) est plus faible que le poids de Hamming de d, ρ a une plus forte probabilité de valoir 1, et inversement. Avec cette astuce, l'algorithme va tendre à sélectionner le cas ayant le plus faible poids de Hamming, c'est-à-dire la branche la plus rapide.

On note seulement que cet algorithme ne peut pas toujours sélectionner la branche la plus rapide, sinon, il deviendrait déterministe et donc facilement attaquable.

Selon un troisième mode de réalisation de l'invention, on choisit au début du procédé un nombre u aléatoire de v bits et on mémorise x^u dans le registre R1. De préférence, le nombre u est modifié plusieurs fois au cours du procédé, pour augmenter le facteur aléatoire dans le procédé.

Puis, au cours du calcul, pour un rang i(j) donné, on se demande, pour un paquet w de v bits de d tel que w>u, si le calcul de x^w est plus coûteux (en terme de temps de calcul) que celui de x^(w-u)*x^u.

Pour répondre à cette question, il suffit de déterminer si H(w) > H(w-u) + 1. H(w) est le poids de Hamming de w, il est représentatif du coût de l'opération x^w. H(w-u) est le poids de Hamming de x^(w-u), représentatif de x^(w-u). Le terme " + 1" est représentatif du coût de la multiplication de x^(w-u) par x^u (x^u étant mémorisé par ailleurs).

Si le calcul de x^w est moins coûteux que le calcul de x^(w-u)*x^u, alors on continue le procédé. Sinon, si le calcul de x^w est plus coûteux que le calcul de x^(w-u)*x^u, alors on remplace le paquet w de bits de d par le nombre w-u. L'étape de consolidation (ici, une multiplication par x^u, qui ce traduit par l'opération R0 <- R0×R1 mod N) sera réalisée lorsque tous les bits de d modifiés auront été utilisés.

Par rapport aux deux précédents modes de réalisation décrits, ce troisième mode de réalisation présente l'avantage d'être plus rapide, puisque, pour effectuer une randomisation, on choisit chaque fois le chemin le plus rapide (le moins coûteux). Ainsi, on montre expérimentalement que la complexité de ce procédé est d'environ 1.4. La complexité est le nombre moyen de multiplications de contenus de registres réalisées pour chaque bit de l'exposant d. La complexité d'un algorithme SAM non protégé est de 1.5 ; la complexité des procédés selon le premier ou le deuxième modes de réalisation de l'invention est quant à elle légèrement supérieure à 1.5.

Par ailleurs, dans ce troisième mode de réalisation, la source d'aléatoirité (le nombre u) est extérieure au procédé. Enfin, les ressources (notamment le nombre de registres) utilisées sont les mêmes.

Ce troisième mode de réalisation peut être concrétisé par l'algorithme IV suivant :

```
Entrée : x, d = (d_m, ..., d_0)_2
Paramètres : v, k
Sortie : y = x^d mod N
  R0 <- 1 ; R2 <- x ; i <- m ; L = {}
  tant que i ≥ 0, faire
      R0 <- R0×R0 mod N
      si d_i = 1 alors R0 <- R0×R2 mod N fin si
      si i = m mod((m+1)/k)) alors σ <- 1 fin si
      si σ = 1 et L = {} alors
          (modification du nombre u en cours de procédé)
          σ <- 0; u <- R{0, …, 2^v-1} ;
          R1 = x^u mod N
      fin si
      w <- d_{i->i-v+1}
      h <- H(w)
      si w ≥ u  alors Δ <- w-u ; h_Δ <- 1 + H(Δ)
                sinon h_Δ <- v+2
      fin si
      ρ <- R{0, 1}
      si [(σ=0)∧(i-v+1≥0)] ∧
              [(h>h_Δ)OU((ρ=1)ET(h=h_Δ))] alors
          (on choisit de réaliser x^(w-u))
              d_{i->i-v+1} <- Δ ; L <- L ∪ {i-v+1}
      fin si
      si (i ∈ L) alors
          R0 <- R0×R1 mod N
          L <- L \{i}
      fin si
      i <- i-1
  fin tant que
retourner R0
```

Dans cet exemple, l'ensemble L contient la liste des indices pour lesquels une étape de consolidation doit être réalisée. L'instruction "si d_i = 1 alors R0 <- R0×R2 mod N fin si" est l'instruction classique d'un algorithme SAM, réalisée pour chaque valeur de i.

L'exposant d est ici découpé en k blocs, de taille identique si m+1 est divisible par k ou de taille identique à une unité près sinon.

A chaque début de bloc (c'est à dire pour i = m mod((l+1)/k)), on met la variable σ à 1. Ensuite, quand σ vaut 1, il faut attendre que l'ensemble L soit vide avant d'effectuer une nouvelle étape de randomisation. A chaque étape de consolidation, on enlève un indice i(j) correspondant de l'ensemble L (instruction L <- L \{i}). Lorsque l'ensemble L est vidé, une nouvelle valeur de u peut être choisie et x^u est calculé par un algorithme SAM classique utilisant les registres R1 et R2.

Au milieu de chaque bloc (σ = 0), on réalise une ou plusieurs étapes de randomisation, lorsque $h>h_\Delta$ ou (ρ=1 et $h = h_\Delta$), et on mémorise à chaque fois (instruction L <- $L \cup$ {i-v+1}) l'indice i(j)-v+1 auquel on devra effectuer une étape de consolidation. Il faut donc que i-v+1 soit un indice de consolidation valable, c'est-à-dire que i-v+1 ≥0 (condition de consistance). A chaque étape de randomisation, si $h > h_\Delta$, on choisit de réaliser l'opération x^(w-u)*x^u, moins coûteuse, et on modifie les bits de d en conséquence($d_{i->i-v+1}$ <- Δ). Si $h < h_\Delta$, on choisit de réaliser x^w, moins coûteuse, et on ne modifie pas d. Si $h = h_\Delta$, on choisit aléatoirement (ρ = 0 ou 1 aléatoire) de réaliser x^(w-u)*x^u ou x^w.

A titre indicatif, on donne le nombre moyen de multiplications modulaires nécessaires pour réaliser une exponentiation de longueur 1024 avec l'algorithme SAM, protégé ou non :

- SAM sans protection : 1536 multiplication
- SAM protégé en ajoutant un multiple de $\Phi(n)$ ($r.\Phi(n)$ avec r de 64 bits) ajouté à l'exposant d (art antérieur) : 1536 + 96 multiplications
- SAM protégé selon l'algorithme II ou III : 1536 + 10 multiplications
- SAM protégé selon l'algorithme I' : 1536+512 multiplications. $\overline{\rho}$, $\overline{\rho}$ étant la valeur moyenne de ρ
- SAM protégé selon l'algorithme IV : 1443 multiplications

On constate par ces exemples qu'un algorithme protégé selon l'invention est très efficace, en terme de multiplications réalisées (et donc de temps de calcul).

REVENDICATIONS

1. Procédé cryptographique dans un composant électronique au cours duquel on réalise une exponentiation modulaire de type x^d, avec d un exposant entier de m+1 bits, en balayant les bits de d de gauche à droite dans une boucle indicée par i variant de m à 0 et en calculant et en mémorisant dans un accumulateur (R0), à chaque tour de rang i, un résultat partiel actualisé égal à x^b(i), b(i) étant les m-i+1 bits de poids les plus forts de l'exposant d (b(i) = $d_{m->i}$),

le procédé étant caractérisé en ce que, à la fin d'un tour de rang i(j) (i = i(0)) choisi aléatoirement, on réalise une étape E1 de randomisation au cours de laquelle :

E1 : on soustrait un nombre z (z= b(i(j)), z = b(i(j)).2^{τ}, z = u) aléatoire à une partie des bits de d non encore utilisés ($d_{i-1->0}$) dans le procédé

puis, après avoir utilisé les bits de d modifiés par l'étape de randomisation E1, on réalise une étape de consolidation E2 au cours de laquelle :

E2 : on mémorise (R0 <- R1×R0) dans l'accumulateur (R0) le résultat de la multiplication du contenu de l'accumulateur (x^b(i)) par un nombre fonction de x^z mémorisé dans un registre (R1).

2. Procédé selon la revendication précédente, dans lequel l'étape E1 est répétée une ou plusieurs fois, à la fin de différents tours de rang i(j) (i = i(0), i = i(1), ...) choisis aléatoirement entre 0 et m.

3. Procédé selon la revendication précédente, dans lequel, à chaque tour i, on décide aléatoirement (ρ=1) si on réalise l'étape E1 ou pas.

4. Procédé cryptographique selon l'une des revendications 1 à 3, dans lequel le nombre z (z= b(i(j)), z = b(i(j)).2^{τ}) est fonction de l'exposant d, dans lequel, lors de l'étape de randomisation, on mémorise (R1 <- R0×R1) également dans un registre (R1) le résultat de la multiplication du contenu de l'accumulateur (x^b(i)) par le contenu du dit registre (R1).

5. Procédé selon la revendication 4, dans lequel l'étape de consolidation E2 est réalisée après le dernier tour de rang i égal à 0.

6. Procédé selon la revendication précédente, au cours duquel, lors de l'étape E1, on soustrait à d le nombre b(i).

7. Procédé selon la revendication 6, au cours duquel on réalise :

Entrée : x, d = $(d_m, \ldots, d_0)_2$

Sortie : y = x^d mod N

R0 <- 1 ; R1 <- 1 ; R2 <- x ; i <- m

tant que i ≥ 0, faire :

R0 <- R0×R0 mod N

si d_i = 1 alors R0 <- R0×R2 mod N

ρ <- R{0, 1}

si ((ρ = 1) ET $d_{i-1 \to 0} \geq d_{m-i}$ alors

$d \leftarrow d - d_{m->i}$

R1 <- R1×R0 mod N

fin si

i <- i-1

fin tant que

R0 <- R0×R1 mod N

retourner R0

8. Procédé selon la revendication 5, au cours duquel l'étape E1 est modifiée comme suit :

E1 : on soustrait à d un nombre égal à g.b(i), g étant un nombre entier positif ; on élève le résultat partiel actuel (x^b(i)) à la puissance g et on mémorise le résultat dans le registre (R1).

9. Procédé selon la revendication précédente, dans lequel g est égal à 2^{τ}, τ étant un nombre aléatoire choisi entre 0 et T.

10. Procédé selon la revendication précédente, dans lequel on réalise :

```
Entrée : x, d = (d_m, ..., d_0)_2
Sortie : y = x^d mod N
    R0 <- 1 ; R1 <- 1 ; R2 <- x ; i <- m
    tant que i ≥ 0, faire :
        R0 <- R0×R0 mod N
        si d_i = 1 alors R0 <- R0×R2 mod N
        ρ <- R{0, 1} ; τ <- R{0, ..., T}
        si ((ρ = 1) ET (d_{i-1->τ} ≥ d_{m-i})) alors
            d_{i-1->τ} ← d_{i-1->τ} − d_{m->i}
            R3 <- R0
            tant que (τ > 0) faire
                R3 <- R3^2 mod N ; τ <- τ-1
            fin tant que
            R1 <- R1×R3 mod N
        fin si
        i <- i-1
    fin tant que
    R0 <- R0×R1 mod N
retourner R0
```

11. Procédé selon l'une des revendications 1 à 4, dans lequel l'étape de consolidation E2 est réalisée à la fin du rang utilisant le dernier bit de d modifié lors de l'étape E1.

12. Procédé selon la revendication 11, au cours duquel, lors de l'étape E1, on soustrait le nombre b(i) aux bits de d de rang i(j) - c(j) à i(j)-1, c(j) étant un nombre entier et on mémorise le contenu de l'accumulateur (x^b(i(j)) dans le registre (R1).

13. Procédé selon la revendication précédente, au cours duquel, lors du tour de rang i(j+1), on choisit

aléatoirement de réaliser l'étape E1 uniquement si $i(j+1) \leq i(j)-c(j)$. ($\sigma = 1$ sémaphore libre).

14. Procédé selon la revendication 12 ou 13, dans lequel $c(j)$ est égal à $m-i(j)+1$.

15. Procédé selon la revendication précédente, au cours duquel on réalise les étapes suivantes :

Entrée : x, $d = (d_m, \ldots, d_0)_2$

Sortie : $y = x^d \bmod N$

R0 <- 1 ; R1 <- 1 ; R2 <- x ;
i <- m ; c <- -1 ; σ <- 1
tant que $i \geq 0$, faire :
R0 <- R0×R0 mod N
si $d_i = 1$ alors R0 <- R0×R2 mod N fin si
si (2i ≥ m+1) ET (σ = 1) alors c <- m-i+1
sinon σ = 0
fin si
ρ <- R{0, 1}
ε <- ρ ET ($d_{i-1 \rightarrow i-c} \geq d_{m \rightarrow i}$) ET σ
si ε = 1 alors
R1 <- R0 ; σ <- 0
$d_{i-1 \rightarrow i-c} \leftarrow d_{i-1 \rightarrow i-c} - d_{m \rightarrow i}$
fin si
si c = 0 alors
R0 <- R0×R1 mod N ; σ <- 1
fin si
c <- c-1 ; i <- i-1
fin tant que
retourner R0

16. Procédé selon la revendication 12 ou 13, dans lequel $c(j)$ est choisi aléatoirement entre $i(j)$ et $m-i(j)+1$.

17. Procédé selon la revendication précédente, au cours duquel on réalise :

```
Entrée : x, d = (d_m, ..., d_0)_2
Sortie : y = x^d mod N
    R0 <- 1 ; R1 <- 1 ; R2 <- x ;
    i <- m ; c <- -1 ; σ <- 1
    tant que i ≥ 0, faire :
        R0 <- R0×R0 mod N
        si d_i = 1 alors R0 <- R0×R2 mod N
            si (2i ≥ m+1) ET (σ = 1)
                    alors c <- R{m-i+1, ..., i}
                    sinon σ = 0
            ε <- ρ ET (d_{i-1 -> i-c} ≥ d_{m->i}) ET σ
            si ε = 1 alors
                  R1 <- R0 ; σ <- 0
                  d_{i-1 -> i-c} <- d_{i-1 -> i-c} - d_{m -> i}
            fin si
            si c = 0 alors
                R0 <- R0×R1 mod N ; σ <- 1
            fin si
            c <- c-1 ; i <- i-1
    fin tant que
retourner R0
```

18. Procédé selon l'une des revendications 1 à 2, dans lequel le nombre z est un nombre u (z = u) de v bits choisi aléatoirement et indépendant de l'exposant d.

19. Procédé selon la revendication précédente, dans lequel, au cours de l'étape E1, le nombre u est soustrait à un paquet w de v bits de d.

20. Procédé selon la revendication précédente, au cours duquel :

- si H(w-u) + 1 < H(w), on choisit de réaliser une étape E1 de randomisation,

- si H(w-u) + 1 > H(w), on choisit de ne pas réaliser d'étape E1,
- si H(w-1) + 1 = H(w), on choisit aléatoirement de réaliser ou non une étape E1 de randomisation.

21. Procédé selon la revendication précédente, au cours duquel on réalise :

```
Entrée : x, d = (d_m, ..., d_0)_2
Paramètres : v, k
Sortie : y = x^d mod N
  R0 <- 1 ; R2 <- x ; i <- m ; L = {}
  tant que i ≥ 0, faire
      R0 <- R0×R0 mod N
      si d_i = 1 alors R0 <- R0×R2 mod N fin si
      si i = m mod((m+1)/k)) alors σ <- 1 fin si
      si σ = 1 et L = {} alors
          s <- 0; u <- R{0, …, 2^v-1} ;
          R1 = x^u mod N
      fin si
      w <- d_{i->i-v+1}
      h <- H(w)
      si w ≥ u  alors Δ <- w-u ; h_Δ <- 1 + H(Δ)
                sinon h_Δ <- v+2
      fin si
      ρ <- R{0, 1}
      si [(σ=0)∧(i-v+1≥0)] ∧
              [(h>h_Δ)OU((ρ=1)ET(h=h_Δ))] alors
              d_{i->i-v+1} <- Δ ; L <- L ∪ {i-v+1}
      fin si
      si (i ∈ L) alors
          R0 <- R0×R1 mod N
          L <- L \{i}
      fin si
      i <- i-1
  fin tant que
retourner R0
```